平衡树

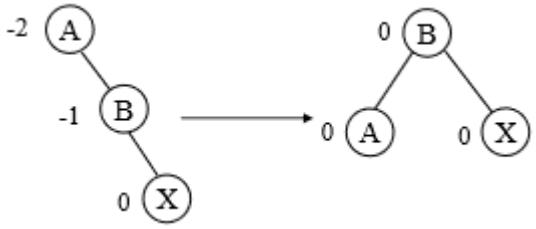
2019年6月5日　　20:27

-
- 平衡二叉树AVL

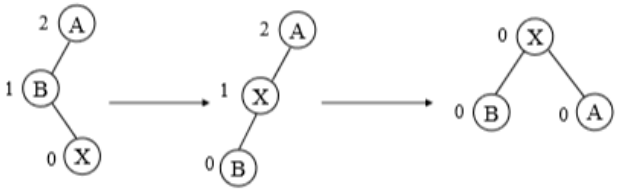
1. 平衡二叉树Balanced Binary Tree 或Height Balanced Tree
    1) AVL树：任何结点的左子树和右子树的深度最多相差**1**的二叉树
        i. 得名于G. M. Adelson-Velsky和E. M. Landis
    2) 平衡因子Balance Factor：该结点的左子树的深度减去右子树的深度，平衡二叉树上所有结点的平衡因子只可能是 - 1、0和1
        i. 每当插入一个结点时，首先，检查是否因插入结点而破坏了树的平衡性，若是，则找出其中最小不平衡子树，在保持排序树特性的前提下，调整最小不平衡 子树中各结点之间的链接关系，以达到新的平衡
2. 旋转
    1) LL旋转平衡(单向左旋) （LL型）　由于在A结点的左子树的左子树中插入结点，导致平衡 因子为2而失去平衡。则需要进行顺时针旋转

    i. 

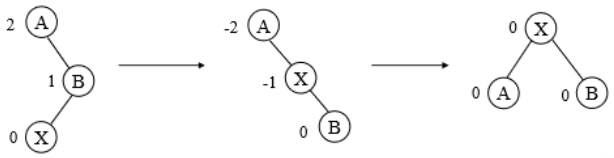
    2) RR旋转平衡(单向右旋) （RR型）　在A的右子树的右子树中插入结点，使A的平衡因子由 - 1变为 - 2而失去平衡。需要进行逆时针旋转

    i. 

    3) LR旋转平衡(先左后右)：　（LR型）　由于在A的左子树的右子树中插入结点，使A的平衡因子 由1增至2而失去平衡，需要两次旋转，先逆时针，后顺时针

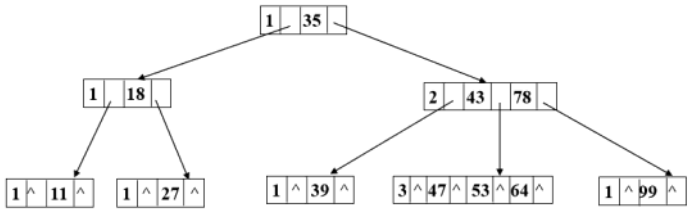
    i. 

    4) RL旋转平衡(先右后左)：　（RL型）　由于在A的右子树的左子树中插入结点，使A的平衡因子 由 - 1增至 - 2而失去平衡，需要进行两次旋转平衡，先顺时针，再逆时针

    i. 

-
- B-树

3. B-树是一种平衡的多路查找树

1) 

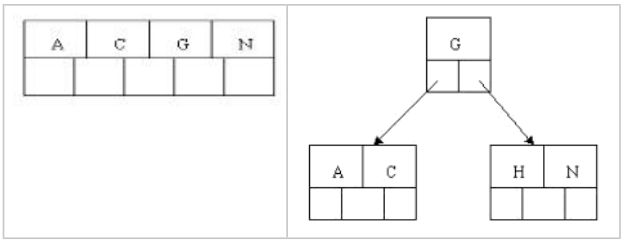
一棵4阶的B-树

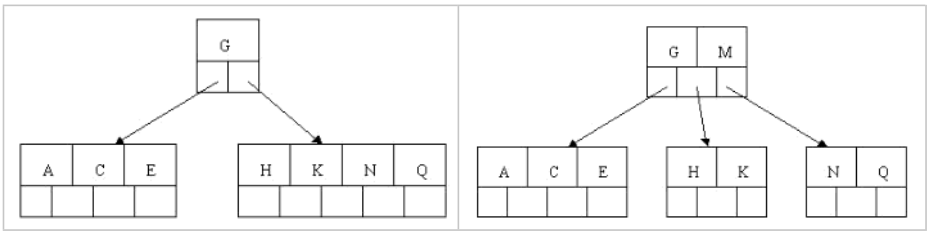
2) 非叶结点都有2或3个子树的B树可称为2-3树
3) 类似的可有2-3-4树，它与红黑树的实现有一定的类似

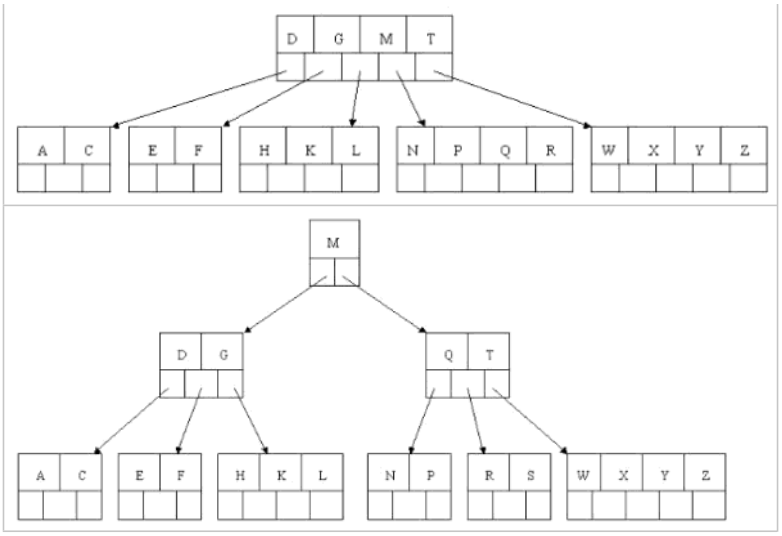
4. 一棵m阶的B-树，或为空树，或为满足以下特性的m叉树：
   1) 树中每个结点**至多m棵子树（<=m）**
   2) 若根结点不是叶子结点，则至少有两棵子树（因为插入并分裂后只能有两棵）
   3) 除根之外的所有非终端结点**至少有上取整m/2棵子树（>=m/2）**
   4) 所有的非终端结点中包含下列信息数据（n, A0, K1, A1, K2, A2, ..., Kn, An）n可省略。其中Ki 为关键字，从小到大；Ai为指向子树根结点的指针。即n-1个关键字对应n个子树指针
   5) 所有的叶子结点都出现在同一层次上

5. B-树的插入
   1) 每插入一个关键字都首先在**最底层**的某个非终端结点中添加一个关键字，若该结点的关键字个数不超过m-1，则插入完成，否则产生“分裂”：
   2) 挑选一个**“中间”值移到上层**，调整中间值左右的结点在该中间值的左右指针，若其上一层也到m了，就再向上分裂
   3) 例：5阶树插入C N G A **H**　E K Q **M**　F W L T Z D P R X Y **S**

来自 <https://blog.csdn.net/v_JULY_v/article/details/6530142>







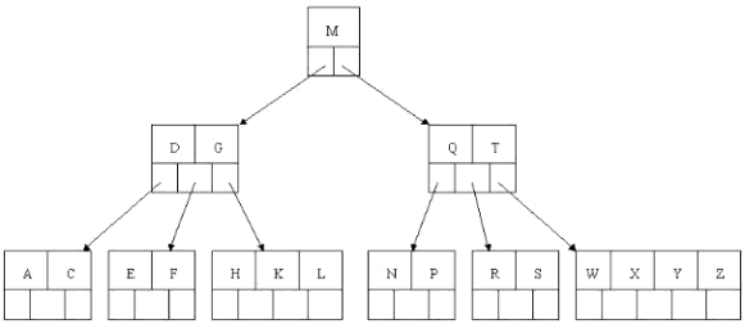
6. B-树的删除

1) 被删关键字所在结点中的关键字数目不小于⌈m/2⌉,则只需删除该关键字和相应指针，其他不变

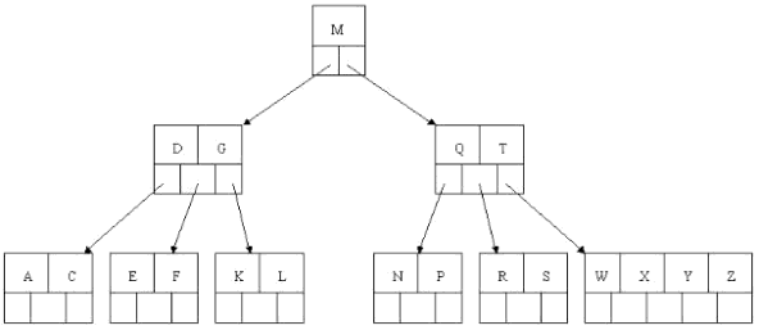
2) 被删关键字所在结点中的关键字数目等于⌈m/2⌉-1, 而与该结点相邻结点中的关键字数目大于⌈m/2⌉-1，则需将其兄弟结点中的最小或最大关键字上移至双亲结点中，再将原双亲结点下移到刚删除的位置

3) 被删关键字所在结点和其相邻兄弟结点中的关键字数目都等于⌈m/2⌉-1。假设该结点有右兄弟，且其右兄弟结点地址由双亲结点中的指针Ai所指，则在删去关键字后，它所在的结点中的剩余的关键字和指针，加上双亲结点中的关键字Ki一起，合并到Ai 所指兄弟结点中（若没有右兄弟，则合并至左兄弟结点中）

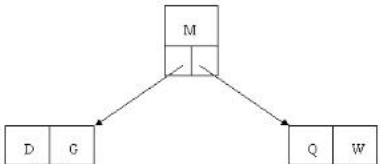
4) 例：5阶B树删除H T R E



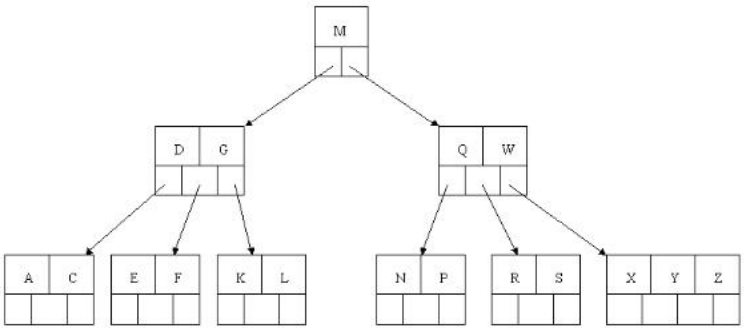
i.



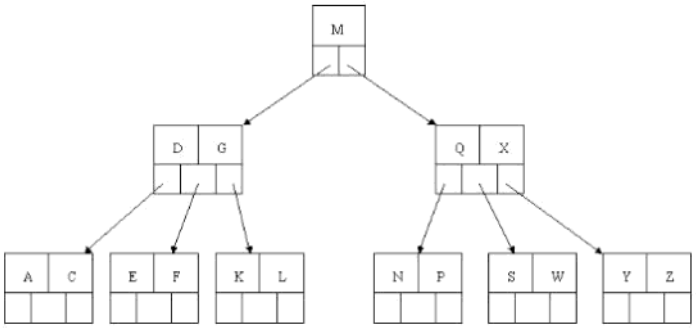
..



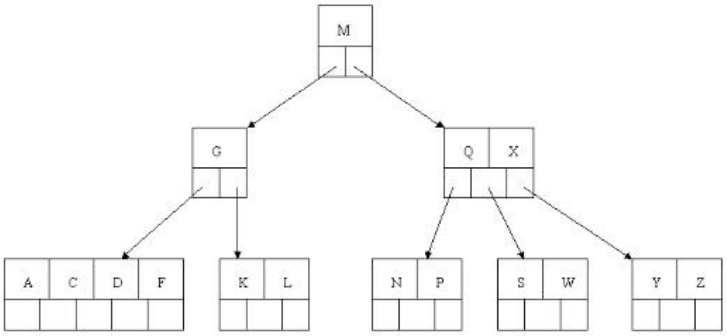
ii.

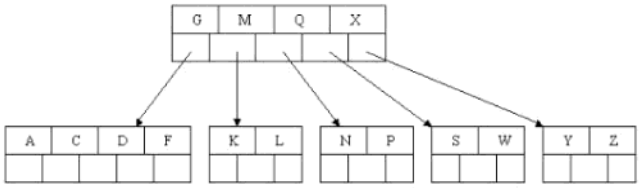


iii.



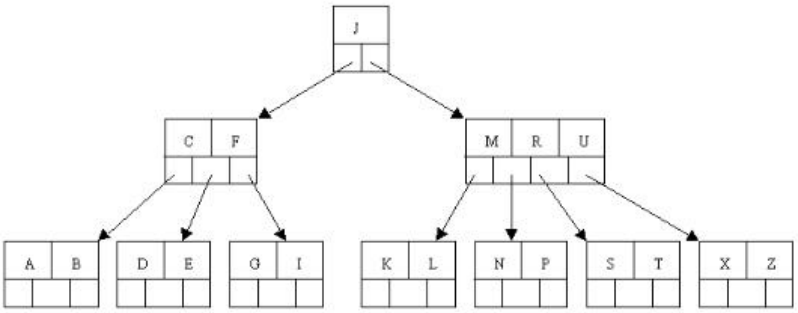
iv.

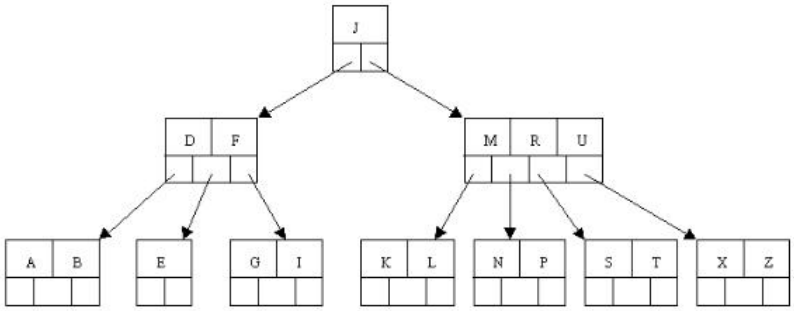


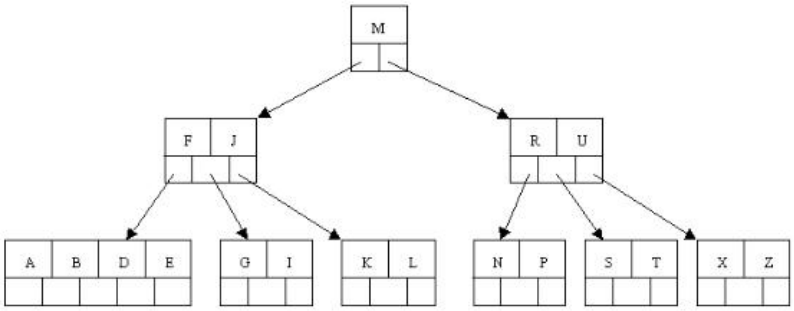


5) 例2：5阶B树删除C

i.







7. 变形
    1) B+树：叶结点包含全部关键字信息及指针，即**非终端节点只做索引功能**，且叶结点一般会按关键字大小顺序再链接起来。似乎允许让n个关键字只对应n个指针，即第一个关键字前面可以没有指针，待考察
    2) B*树：B+树的基础上再让非终端同层结点也用链表指向兄弟结点，且规定非叶结点的关键字个数至少2/3*M个，并重新定义了插删方法
    3) R树：扩展到二维区域分割的B树
8. 2-3-4树
    1) n-节点：n-1个元素，n个子节点
    2) 2-3-4树有2-节点，3-节点，4-节点
    3) 将3-节点拆成黑根红子结点，4-结点拆成黑根红双子结点即实现了红黑树
    4) 由于3-节点的拆分有两种
9. 红黑树
    1) 原理
        i. 每个节点非红即黑；
        ii. 根节点总是黑色的；
        iii. 每个叶子节点都是黑色的空节点（NIL节点）；
        iv. 如果节点是红色的，则它的子节点必须是黑色的（反之不一定）；
        v. 从根节点到叶节点或空子节点的每条路径，必须包含相同数目的黑色节点（即相同的黑色高度）
    2) 把红节点收缩到其父节点，就会变成2-3-4树
    3) 新插入结点是红色的
    4) 不是AVL树，但统计上性能更好
        -
        - 哈希Hash
10. 哈希函数：从关键字集合到地址集合的映射
    1) 直接定值法：取关键字的某个线性函数值

2) 数字分析法：实现知道关键字集合的话，挑选分布较均匀的若干位
3) 平方取中法：挑关键字的平方的中间几位
4) 折叠法：分割出位数相同的几部分，取叠加和
5) 除留取余法：除以一个不大于哈希表长m的数p后的余数，这个p一般是余数或包含不小于20的质因子的合数

11. 哈希表：用函数将关键字直接映射到一个表内对应地址上
    1) 散列/哈希造表：映射过程
    2) 哈希地址/散列地址：映射位置
    3) 各哈希地址被映射的概率相同时，称该哈希函数为uniform均匀的
    4) 哈希函数需考虑的因素
        i. 计算所需时间
        ii. 关键字长度
        iii. 哈希表大小
        iv. 关键字分布情况
        v. 记录查找频率
12. 冲突collision：不同关键字在某一哈希函数后得到某一哈希地址
    1) 同义词synonym：具有同函数值的关键字是该哈希函数的~
    2) 解决冲突的方案
        i. 开放定址线性探测法：求偏移+1的哈希地址（爆表后回到表头）
        ii. 开放定址二次探测法：依次求偏移正负1，2，3……的平方的地址
        iii. 开放定址随机探测法：求偏移伪随机数序列的地址
        iv. 再哈希法：用另一个哈希函数
        v. 链地址法：将同义词存在同一线性链表中